DA_project_final

February 15, 2020

1 Machine Learning aided Record Linkage - a comparison between different methods

1.1 Group Components

- **Francesco Porto** f.porto2@campus.unimib.it (816042)
- **Francesco Stranieri** f.stranieri1@campus.unimib.it (816551)
- **Mattia Vincenzi** m.vincenzi14@campus.unimib.it (860579)

1.2 Abstract

Record Linkage is the process of finding records in one or more datasets that refer to the same entity across different data sources. Traditionally, it is done by applying comparison rules between pairs of attributes from each dataset. In this project we investigate some possible Machine Learning applications to Record Linkage (and Data deduplication), and we compare them to the deterministic approach.

1.3 Python Record Linkage Toolkit

Throughout the project, we make use of a Python library called "Python Record Linkage Toolkit", which provides a simple framework to facilitate the process of Record Linkage. In the context of this library, the Record Linkage process is divided into 5 steps:

- **Preprocessing**: clean the datasets in order to improve the chances of finding matches
- **Indexing**: create pairs of records, called candidate links, from each dataset
- **Comparison**: compare record pairs via different algorithms, thresholds, etc.
- **Classification**: classify record pairs into matches or non-matches
- **Evaluation**: evaluate classification results

1.4 Helper Functions

We have defined a few helper functions that allow us to streamline the Record Linkage process: - **Indexing_dataset** : given two datasets, returns the result of the indexing process (either *full* or *block*) on the given attributes

- **Compare_records** : given two datasets, a set of rules specifying how to compare some attributes, and a list of attributes that must match exactly, returns a multi index containing the comparisons' results.

- **Evaluate_results** : returns a confusion matrix of a given classification method; also returns its precision, f-score and recall

1.5 Chapter 1: Record Linkage

In this chapter we present two examples of Record Linkage, the first one using a **Full Index**, that is we make the cartesian product of the records in the first dataset and the records in the second one; the second one uses **Blocking**, a more optimal approach that only finds a subset of the cartesian product.

1.6 Experiment 1: Using a Full index

1.6.1 Dataset description

We use the FEBRL (Freely Extensible Biomedical Record Linkage) dataset since it provides the *true links* for optimal Record Linkage. The paper used as reference is available here.

```
In [51]: from recordlinkage.datasets import load_febrl4
```

This dataset contains 10000 records (5000 originals and 5000 duplicates, with one duplicate per original); the originals have been split from the duplicates into dataset4a.csv (containing the 5000 original records) and dataset4b.csv (containing the 5000 duplicate records).

```
Loading data...
5000 records in dataset A
5000 records in dataset B
5000 links between dataset A and B
```

We take a first look at the dataset. It contains auto-generated records about patients. There are 11 fields: - **rec_id**: the id of the record - **given_name**: the name of the patient - **street_number**: the street number of the patient's house - **address_1**: the road where the patient lives - **address_2**: the city where the patient lives - **suburb**: the suburb in the city where the patient lives - **postcode**: the postcode of the city where the patient lives - **state**: the state where the patient lives - **date_of_birth**: the date of birth of the patient - **soc_sec_id**: the social security number of the patient

```
             given_name    surname street_number           address_1  \
rec_id
rec-1070-org   michaela    neumann             8      stanley street
rec-1016-org   courtney    painter            12   pinkerton circuit
rec-4405-org    charles      green            38 salkauskas crescent
rec-1288-org    vanessa       parr           905      macquoid place
rec-3585-org    mikayla   malloney            37       randwick road
...                 ...        ...           ...                 ...
rec-2153-org    annabel   grierson            97   mclachlan crescent
rec-1604-org     sienna   musolino            22      smeaton circuit
rec-1003-org    bradley   matthews             2        jondol place
rec-4883-org     brodee       egan            88         axon street
rec-66-org        koula  houweling             3      mileham street

                      address_2            suburb postcode state  \
rec_id
rec-1070-org                miami     winston hills     4223   nsw
rec-1016-org           bega flats         richlands     4560   vic
rec-4405-org                 kela             dapto     4566   nsw
rec-1288-org   broadbridge manor     south grafton     2135    sa
```

```
rec-3585-org             avalind  hoppers crossing     4552   vic
...                          ...               ...      ...   ...
rec-2153-org        lantana lodge           broome     2480   nsw
rec-1604-org             pangani          mckinnon     2700   nsw
rec-1003-org        horseshoe ck       jacobs well     7018    sa
rec-4883-org         greenslopes          wamberal     2067   qld
rec-66-org    old airdmillan road      williamstown     2350   nsw

             date_of_birth soc_sec_id
rec_id
rec-1070-org      19151111    5304218
rec-1016-org      19161214    4066625
rec-4405-org      19480930    4365168
rec-1288-org      19951119    9239102
rec-3585-org      19860208    7207688
...                    ...        ...
rec-2153-org      19840224    7676186
rec-1604-org      19890525    4971506
rec-1003-org      19481122    8927667
rec-4883-org      19121113    6039042
rec-66-org        19440718    6375537

[5000 rows x 10 columns]
```

We notice that the records having the same numeric id represent the same entity.

```
given_name             michaela
surname                 neumann
street_number                 8
address_1        stanley street
address_2                 miami
suburb            winston hills
postcode                   4223
state                       nsw
date_of_birth          19151111
soc_sec_id              5304218
Name: rec-1070-org, dtype: object
```

```
given_name             michafla
surname                 jakimow
```

```
street_number                 8
address_1        stanleykstreet
address_2                 miami
suburb           winstonbhills
postcode                   4223
state                       NaN
date_of_birth          19151111
soc_sec_id              5304218
Name: rec-1070-dup-0, dtype: object
```

1.6.2 Indexing (Full)

Indexing is the process of creating all the possible links, also called **candidate links**, between the two datasets. In this specific example, we use a technique called **Full Indexing**, which returns the cartesian products of the records from each dataset.

```
WARNING:recordlinkage:indexing - performance warning - A full index can result in large number of record

WARNING:recordlinkage:indexing - performance warning - A full index can result in large number of record
```

```
MultiIndex([('rec-1070-org',  'rec-561-dup-0'),
            ('rec-1070-org', 'rec-2642-dup-0'),
            ('rec-1070-org',  'rec-608-dup-0'),
            ('rec-1070-org', 'rec-3239-dup-0'),
            ('rec-1070-org', 'rec-2886-dup-0'),
            ('rec-1070-org', 'rec-4285-dup-0'),
            ('rec-1070-org',  'rec-929-dup-0'),
            ('rec-1070-org', 'rec-4833-dup-0'),
            ('rec-1070-org',  'rec-717-dup-0'),
            ('rec-1070-org', 'rec-3984-dup-0'),
            ...
            (  'rec-66-org',  'rec-670-dup-0'),
            (  'rec-66-org', 'rec-4134-dup-0'),
            (  'rec-66-org', 'rec-3866-dup-0'),
            (  'rec-66-org', 'rec-3152-dup-0'),
            (  'rec-66-org', 'rec-3363-dup-0'),
            (  'rec-66-org', 'rec-4495-dup-0'),
            (  'rec-66-org', 'rec-4211-dup-0'),
            (  'rec-66-org', 'rec-3131-dup-0'),
            (  'rec-66-org', 'rec-3815-dup-0'),
            (  'rec-66-org',  'rec-493-dup-0')],
           names=['rec_id_1', 'rec_id_2'], length=25000000)
```

1.6.3 Comparison

Comparison refers to the process of evaluating all the possible candidate links in order to figure out all the attributes having the same value. In order to compare attributes, we need to specify (for each attribute): * A **metric** to be used for evaluating the similarity between each pair of the same attributes of the different dataset * A **threshold** to decide under which circumstances the metric shall return 1 (the similarity is 1 in case of agreement) or 0 (in case of complete disagreement)

In this specific example we used the Jaro-Winkler method since, according to the documentation, it is faster than the Levenshtein distance and much faster than the Damerau-Levenshtein distance. Since comparing strings is computationally expensive, we chose a subset of attributes on which some errors are permitted.

```
                              surname  address_1  address_2  given_name  \
rec_id_1     rec_id_2
rec-1070-org rec-561-dup-0        0.0        0.0        0.0           0
             rec-2642-dup-0       0.0        0.0        0.0           0
             rec-608-dup-0        0.0        0.0        0.0           0
             rec-3239-dup-0       0.0        0.0        0.0           0
             rec-2886-dup-0       0.0        0.0        0.0           0
...                               ...        ...        ...         ...
rec-66-org   rec-4495-dup-0       0.0        0.0        0.0           0
             rec-4211-dup-0       0.0        0.0        0.0           0
             rec-3131-dup-0       0.0        0.0        0.0           0
             rec-3815-dup-0       0.0        0.0        0.0           0
             rec-493-dup-0        0.0        0.0        0.0           0

                              date_of_birth  suburb  state
rec_id_1     rec_id_2
rec-1070-org rec-561-dup-0                0       0      0
             rec-2642-dup-0               0       0      1
             rec-608-dup-0                0       0      0
             rec-3239-dup-0               0       0      0
             rec-2886-dup-0               0       0      0
...                                     ...     ...    ...
rec-66-org   rec-4495-dup-0               0       0      1
             rec-4211-dup-0               0       0      0
             rec-3131-dup-0               0       0      0
             rec-3815-dup-0               0       0      0
             rec-493-dup-0                0       0      0

[25000000 rows x 7 columns]
```

Note: no cardinality reduction is employed. The output of indexing phase has the same cardinality of the comparison phase. The columns represent the results of the comparison.

1.6.4 Classification

In this phase, candidate links are classified into matches or non-matches. We analyze 3 different classification methods: - **Deterministic approach** (no Machine Learning used) - **Naive Bayes** (Supervised Learning) - **Expectation-Conditional Maximisation** (Unsupervised Learning)

Deterministic approach In the deterministic approach, we consider all the records that agree on at least a certain amount of attributes according to the rules we have previously specified.

```
7.0         1304
6.0         1860
```

```
5.0        1260
4.0         441
3.0         452
2.0       69727
1.0     5634539
0.0    19290417
dtype: int64
```

We notice that 1304 records have 7 common attributes, according to the comparison metrics that we chose. 1860 records have 6 common attributes instead, and so on.

We consider a match all the pairs that agree on at least 4 attributes. In the following table, the matches according to this rule are shown.

```
                             surname  address_1  address_2  given_name  \
rec_id_1     rec_id_2
rec-1016-org rec-1016-dup-0      1.0        1.0        0.0           1
rec-4405-org rec-4405-dup-0      1.0        1.0        1.0           1
rec-1288-org rec-1288-dup-0      1.0        1.0        1.0           1
rec-3585-org rec-3585-dup-0      1.0        1.0        1.0           1
rec-298-org  rec-298-dup-0       1.0        0.0        0.0           1
...                              ...        ...        ...         ...
rec-2153-org rec-2153-dup-0      1.0        0.0        0.0           0
rec-1604-org rec-1604-dup-0      0.0        1.0        1.0           0
rec-1003-org rec-1003-dup-0      0.0        1.0        1.0           0
rec-4883-org rec-4883-dup-0      1.0        1.0        1.0           1
rec-66-org   rec-66-dup-0        1.0        1.0        1.0           1

                             date_of_birth  suburb  state
rec_id_1     rec_id_2
rec-1016-org rec-1016-dup-0              1       1      0
rec-4405-org rec-4405-dup-0              1       1      1
rec-1288-org rec-1288-dup-0              1       0      1
rec-3585-org rec-3585-dup-0              1       1      1
rec-298-org  rec-298-dup-0               1       1      1
...                                    ...     ...    ...
rec-2153-org rec-2153-dup-0              1       1      1
rec-1604-org rec-1604-dup-0              1       1      1
rec-1003-org rec-1003-dup-0              1       0      1
rec-4883-org rec-4883-dup-0              1       1      1
rec-66-org   rec-66-dup-0                1       1      1

[4865 rows x 7 columns]
```

Naive Bayes Naive Bayes is a supervised-learning algorithm that provides a method for computing the probability of a given hypotesis on prior knowledge. It is based on the assumption that all features are conditionally independent.

We initialise the NaiveBayesClassifier.

```
nb_full = rl.NaiveBayesClassifier()
nb_full.fit(features_full, true_links)
```

We print also the parameters that are trained. - p is the probability P(Match) - m is the probability that P(x_i=1|Match) - u is the probability that P(x_i=1|Non-Match)

```
print("p probability P(Match):", nb_full.p)
print()
print("m probabilities P(x_i=1|Match):", nb_full.m_probs)
print()
print("u probabilities P(x_i=1|Non-Match):", nb_full.u_probs)
print()
print("weights of features:", nb_full.weights)
```

```
p probability P(Match): 0.00019999999999999985

m probabilities P(x_i=1|Match): {'surname': {0.0: 0.14700001411999955, 1.0: 0.8529999858800007}, 'addres

u probabilities P(x_i=1|Non-Match): {'surname': {0.0: 0.9951189437847974, 1.0: 0.004881056215203998}, 'a

weights of features: {'surname': {0.0: 0.14772104886367185, 1.0: 174.75725504307687}, 'address_1': {0.0
```

We tried to predict the matches starting by the output of the comparison phase, using the trained model. We know from the documentation that the total matches are 5000 and our model found 4958.

```
Predicted number of links: 4958
```

We predict the match probability for each pair in the dataset, since we apply the Full index method.

```
probs_bayes_full = nb_full.prob(features_full)
probs_bayes_full
```

```
rec_id_1      rec_id_2
rec-1070-org  rec-561-dup-0     5.362010e-10
              rec-2642-dup-0    3.086106e-08
              rec-608-dup-0     5.362010e-10
              rec-3239-dup-0    5.362010e-10
              rec-2886-dup-0    5.362010e-10
                                    ...
rec-66-org    rec-4495-dup-0    3.086106e-08
              rec-4211-dup-0    5.362010e-10
              rec-3131-dup-0    5.362010e-10
              rec-3815-dup-0    5.362010e-10
              rec-493-dup-0     5.362010e-10
Length: 25000000, dtype: float64
```

Expectation-Conditional Maximisation Unsupervised learning with the ECM algorithm. This classifier doesn't need training data since it is often hard to collect.

We initialise the Expectation-Conditional Maximisation classifier.

```
ecm_full = rl.ECMClassifier()
ecm_full.fit(features_full)
```

We print the parameters that are trained (m, u and p).

```
p probability P(Match): 0.00019964038993562045

m probabilities P(x_i=1|Match): {'surname': {0.0: 0.14559845395438062, 1.0: 0.8544015460456198}, 'addres

u probabilities P(x_i=1|Non-Match): {'surname': {0.0: 0.9951189185968193, 1.0: 0.0048810814031804844},

weights of features: {'surname': {0.0: 0.1463126177519403, 1.0: 175.04349456022118}, 'address_1': {0.0:
```

```
Predicted number of links: 4958
```

```
rec_id_1      rec_id_2
rec-1070-org  rec-561-dup-0     5.170889e-10
              rec-2642-dup-0    2.958100e-08
              rec-608-dup-0     5.170889e-10
              rec-3239-dup-0    5.170889e-10
              rec-2886-dup-0    5.170889e-10
                                    ...
rec-66-org    rec-4495-dup-0    2.958100e-08
              rec-4211-dup-0    5.170889e-10
              rec-3131-dup-0    5.170889e-10
              rec-3815-dup-0    5.170889e-10
              rec-493-dup-0     5.170889e-10
Length: 25000000, dtype: float64
```

1.6.5 Evaluation

We compare the results of the deterministic method and the two ML-based ones (Naive Bayes and ECM)

```
Deterministic approach

Confusion matrix:
 [[    4865      135]
 [       0 24995000]]
Fscore: 0.9863152559553979
Recall: 0.973
Precision: 1.0
```

```
Naive Bayes

Confusion matrix:
 [[    4954       46]
 [       4 24994996]]
Fscore: 0.9949789114279977
Recall: 0.9908
Precision: 0.9991932230738201
```

```
Expectation-Conditional Maximisation

Confusion matrix:
 [[    4954       46]
 [       4 24994996]]
Fscore: 0.9949789114279977
Recall: 0.9908
Precision: 0.9991932230738201
```

1.7 Experiment 2 : Using a Block index

1.7.1 Indexing (Block)

Indexing is the process of creating all the possible links between the two datasets. In this specific example, we use a technique called **Blocking**, which groups together all the records that agree on AT LEAST one of the specified attributes.

We have decided to use **surname**, **date_of_birth** and **soc_sec_id** as discriminating variables. By doing that, the resulting records will agree on at least one of these attributes.

Note: the cardinality using the Block index method is way less than the Full index method.

```
MultiIndex([(  'rec-0-org',    'rec-0-dup-0'),
            (  'rec-0-org', 'rec-1505-dup-0'),
            (  'rec-0-org', 'rec-1636-dup-0'),
            (  'rec-0-org', 'rec-2074-dup-0'),
            (  'rec-0-org', 'rec-2683-dup-0'),
            (  'rec-0-org', 'rec-2724-dup-0'),
            (  'rec-0-org', 'rec-2894-dup-0'),
            (  'rec-1-org',    'rec-1-dup-0'),
            (  'rec-1-org', 'rec-1052-dup-0'),
            (  'rec-1-org', 'rec-2552-dup-0'),
            ...
            ('rec-999-org', 'rec-3685-dup-0'),
            ('rec-999-org',  'rec-370-dup-0'),
            ('rec-999-org', 'rec-3766-dup-0'),
            ('rec-999-org', 'rec-3862-dup-0'),
            ('rec-999-org', 'rec-3913-dup-0'),
            ('rec-999-org', 'rec-3940-dup-0'),
            ('rec-999-org', 'rec-4941-dup-0'),
            ('rec-999-org',  'rec-859-dup-0'),
            ('rec-999-org',  'rec-911-dup-0'),
```

```
            ('rec-999-org',  'rec-999-dup-0')],
           names=['rec_id_1', 'rec_id_2'], length=87132)
```

We verify that a pair of candidate links agree on the given attributes:

```
given_name               rachael
surname                     dent
street_number                  1
address_1            knox street
address_2        lakewood estate
suburb                    byford
postcode                    4129
state                        vic
date_of_birth           19280722
soc_sec_id               1683994
Name: rec-0-org, dtype: object
```

```
given_name               rachael
surname                     dent
street_number                  4
address_1            knox street
address_2        lakewood estate
suburb                    byford
postcode                    4129
state                        vic
date_of_birth           19280722
soc_sec_id               1683994
Name: rec-0-dup-0, dtype: object
```

1.7.2 Comparison

We used the same metrics and threshold for each pair of attributes used in the previous experiment.

```
                            surname  address_1  address_2  given_name  \
rec_id_1    rec_id_2
rec-0-org   rec-0-dup-0         1.0        1.0        1.0           1
            rec-1505-dup-0      1.0        0.0        0.0           0
            rec-1636-dup-0      1.0        0.0        0.0           0
            rec-2074-dup-0      1.0        0.0        0.0           0
            rec-2683-dup-0      1.0        0.0        0.0           0
...                             ...        ...        ...         ...
rec-999-org rec-3940-dup-0      1.0        0.0        0.0           0
            rec-4941-dup-0      1.0        0.0        0.0           0
            rec-859-dup-0       1.0        0.0        0.0           0
            rec-911-dup-0       1.0        0.0        0.0           0
            rec-999-dup-0       1.0        1.0        1.0           1

                            date_of_birth  suburb  state
rec_id_1    rec_id_2
rec-0-org   rec-0-dup-0                 1       1      1
```

```
            rec-1505-dup-0              0       0      0
            rec-1636-dup-0              0       0      0
            rec-2074-dup-0              0       0      1
            rec-2683-dup-0              0       0      0
...                                   ...     ...    ...
rec-999-org rec-3940-dup-0              0       0      0
            rec-4941-dup-0              0       0      0
            rec-859-dup-0               0       0      0
            rec-911-dup-0               0       0      1
            rec-999-dup-0               1       1      1

[87132 rows x 7 columns]
```

1.7.3 Classification

Deterministic approach Also in this case, we consider a match all the pairs that agree on at least 4 attributes.

```
7.0     1304
6.0     1857
5.0     1257
4.0      438
3.0      268
2.0    18304
1.0    63704
dtype: int64
```

Naive Bayes

```
# Initialise the NaiveBayesClassifier.
nb_block = rl.NaiveBayesClassifier()
nb_block.fit(features_block, true_links)
```

```
p probability P(Match): 0.05726943028967549

m probabilities P(x_i=1|Match): {'surname': {0.0: 0.1462925993469899, 1.0: 0.8537074006530102}, 'addres

u probabilities P(x_i=1|Non-Match): {'surname': {0.0: 0.0077305166474385435, 1.0: 0.9922694833525615},

weights of features: {'surname': {0.0: 18.924039106165434, 1.0: 0.8603584157084078}, 'address_1': {0.0:
```

```
Predicted number of links: 5107
```

```
rec_id_1     rec_id_2
rec-0-org    rec-0-dup-0       1.000000e+00
             rec-1505-dup-0    9.443910e-07
             rec-1636-dup-0    9.443910e-07
             rec-2074-dup-0    5.426506e-05
             rec-2683-dup-0    9.443910e-07
                                   ...
rec-999-org  rec-3940-dup-0    9.443910e-07
             rec-4941-dup-0    9.443910e-07
             rec-859-dup-0     9.443910e-07
             rec-911-dup-0     5.426506e-05
             rec-999-dup-0     1.000000e+00
Length: 87132, dtype: float64
```

Expectation-Conditional Maximisation

```
# Initialise the Expectation-Conditional Maximisation classifier.
ecm_block = rl.ECMClassifier()
ecm_block.fit(features_block)
```

```
p probability P(Match): 0.05890218346289975

m probabilities P(x_i=1|Match): {'surname': {0.0: 0.17208594081201783, 1.0: 0.8279140591879811}, 'addres

u probabilities P(x_i=1|Non-Match): {'surname': {0.0: 0.00587574265639725, 1.0: 0.9941242573436027}, 'ad

weights of features: {'surname': {0.0: 29.287521744108087, 1.0: 0.832807421277747}, 'address_1': {0.0: 0
```

```
Predicted number of links: 5110
```

```
rec_id_1     rec_id_2
rec-0-org    rec-0-dup-0       1.000000
             rec-1505-dup-0    0.000002
             rec-1636-dup-0    0.000002
             rec-2074-dup-0    0.000079
             rec-2683-dup-0    0.000002
                                 ...
rec-999-org  rec-3940-dup-0    0.000002
             rec-4941-dup-0    0.000002
             rec-859-dup-0     0.000002
             rec-911-dup-0     0.000079
             rec-999-dup-0     1.000000
Length: 87132, dtype: float64
```

1.7.4 Evaluation

```
Deterministic approach

Confusion matrix:
 [[ 4856   144]
 [    0 82132]]
Fscore: 0.9853896103896103
Recall: 0.9712
Precision: 1.0



Naive Bayes

Confusion matrix:
 [[ 4956    44]
 [  151 81981]]
Fscore: 0.9807064410804394
Recall: 0.9912
Precision: 0.9704327393773252



Expectation-Conditional Maximisation

Confusion matrix:
 [[ 4959    41]
 [  151 81981]]
Fscore: 0.9810089020771513
Recall: 0.9918
Precision: 0.9704500978473581
```

1.8 Results Analysis

From the previous two experiments, we noticed that: - The cardinality of the candidate links with Full index method is 25000000, with the Block index is 87132. - This leads to a significative difference in the execution time: - The Full index takes 160 seconds circa for the comparison phase; the Block index takes about 0.56 seconds instead. - Naive Bayes takes about 150s with Full index and 0.23s with Block index. - ECM takes about 92s with Full index and 0.29s with Block index. - This does not lead to a significative difference in the evaluation results: - The Full index method brings a very small increase in the Fscore.

We do not consider this result to be meaningful. Blocking should always be preferred, provided that it is clear what attributes should be used in the index.

1.9 Chapter 2: Data Deduplication

Data Deduplication is the process of linking a dataset to itself, in order to find duplicate records.

1.10 Experiment 3 : Creating prediction models

In this experiment, we first train our models on a smaller dataset. Then, we test the models on a bigger one with the *same* attributes but with different distribution of duplicates.

1.11 Part 1 : Training

1.11.1 Dataset description

This data set contains 5000 records (4000 originals and 1000 duplicates), with a maximum of 5 duplicates based on one original record (and a poisson distribution of duplicate records). Distribution of duplicates: 19 originals records have 5 duplicate records 47 originals records have 4 duplicate records 107 originals records have 3 duplicate records 141 originals records have 2 duplicate records 114 originals records have 1 duplicate record 572 originals records have no duplicate record.

```
Loading data...
5000 records in dataset A
1934 links between dataset A and A
```

```
              given_name    surname street_number         address_1  \
rec_id
rec-2778-org       sarah      bruhn            44     forbes street
rec-712-dup-0      jacob     lanyon             5        milne cove
rec-1321-org     brinley  efthimiou            35  sturdee crescent
rec-3004-org     aleisha     hobson            54     oliver street
rec-1384-org       ethan    gazzola            49    sheaffe street
...                  ...        ...           ...               ...
rec-1487-org      thomas      green            44     tuthill place
rec-1856-org       james    mcneill            42  archibald street
rec-3307-org       paige       lock             7  a'beckett street
rec-227-org      antonio    collier            25      govett place
rec-1143-org       harry       ryan            30  van raalte place

                address_2              suburb postcode state date_of_birth  \
rec_id
rec-2778-org   wintersloe        kellerberrin     4510   vic      19300213
rec-712-dup-0     wellwod  beaconsfield upper     2602   vic      19080712
rec-1321-org    tremearne         scarborough     5211   qld      19940319
rec-3004-org    inglewood           toowoomba     3175   qld      19290427
rec-1384-org   bimby vale          port pirie     3088    sa      19631225
...                   ...                 ...      ...   ...           ...
rec-1487-org   holmeleigh         bonny hills     4740   vic      19420210
rec-1856-org          NaN          evans head     2250   nsw      19011207
rec-3307-org      camboon      carina heights     2290   nsw      19871002
rec-227-org     the rocks         broken hill     2304   qld      19400225
rec-1143-org        anana        preston west     2572    wa      19250425

              soc_sec_id
rec_id
rec-2778-org     7535316
rec-712-dup-0    9497788
rec-1321-org     6814956
rec-3004-org     5967384
rec-1384-org     3832742
...                  ...
rec-1487-org     9334580
```

```
rec-1856-org     4837378
rec-3307-org     5142242
rec-227-org      3973395
rec-1143-org     6392122

[5000 rows x 10 columns]
```

We notice that the records having the same numeric id represent the same entity but with some errors in various attributes.

```
given_name                     jacob
surname                       lanyon
street_number                      5
address_1                 milne cove
address_2                   wellwood
suburb            beaconsfield upper
postcode                        2602
state                            vic
date_of_birth               19080712
soc_sec_id                   9497788
Name: rec-712-org, dtype: object
```

```
given_name                     jacob
surname                       lanyon
street_number                      5
address_1                 milne cove
address_2                    wellwod
suburb            beaconsfield upper
postcode                        2602
state                            vic
date_of_birth               19080712
soc_sec_id                   9497788
Name: rec-712-dup-0, dtype: object
```

```
given_name                     jacob
surname                       lanyln
street_number                      5
address_1                 milne cove
address_2                   wellwood
suburb            beaconsfieod upper
postcode                        2602
state                            vic
date_of_birth               19080712
soc_sec_id                   9497788
Name: rec-712-dup-1, dtype: object
```

1.11.2 Indexing (Block)

We have decided to use **surname**, **date_of_birth** and **soc_sec_id** as discriminating variables. It's important to notice that the discriminating variables are the same throughout different experiments.

```
MultiIndex([(    'rec-0-org', 'rec-3741-dup-0'),
            (    'rec-0-org',   'rec-3741-org'),
            (   'rec-10-org',   'rec-2162-org'),
            (   'rec-10-org',    'rec-343-org'),
            (   'rec-10-org',    'rec-956-org'),
            ('rec-100-dup-0',    'rec-100-org'),
            ('rec-100-dup-1',  'rec-100-dup-0'),
            ('rec-100-dup-1',    'rec-100-org'),
            ('rec-100-dup-2',  'rec-100-dup-0'),
            ('rec-100-dup-2',  'rec-100-dup-1'),
            ...
            (  'rec-998-org',   'rec-1578-org'),
            (  'rec-998-org',   'rec-2112-org'),
            (  'rec-998-org',   'rec-2273-org'),
            (  'rec-998-org',   'rec-2670-org'),
            (  'rec-998-org',    'rec-305-org'),
            (  'rec-998-org',    'rec-345-org'),
            (  'rec-998-org',   'rec-3803-org'),
            (  'rec-998-org',   'rec-3936-org'),
            (  'rec-998-org',    'rec-622-org'),
            (  'rec-999-org',   'rec-3297-org')],
           names=['rec_id_1', 'rec_id_2'], length=50929)
```

We show a pair of candidate links in order to verify that at least one of the values of the discriminating variables are equal.

```
In [100]: df.loc[candidate_links[0][0]]
```

```
Out[100]: given_name              annabelle
          surname                  friswell
          street_number                 205
          address_1            meares place
          address_2        sunningdale farm
          suburb              wildes meadow
          postcode                     7018
          state                          wa
          date_of_birth            19761129
          soc_sec_id                9016980
          Name: rec-0-org, dtype: object
```

```
In [101]: df.loc[candidate_links[0][1]]
```

```
Out[101]: given_name            michael
          surname              shepherd
          street_number              70
          address_1        hurry nplace
          address_2            tintagel
          suburb                penguin
          postcode                 4179
```

```
state                        vic
date_of_birth           19761129
soc_sec_id               8270855
Name: rec-3741-dup-0, dtype: object
```

1.11.3 Comparison

Also in this case, we used the same metrics and threshold used in the previous experiment.

```
                             surname  address_1  address_2  given_name  \
rec_id_1    rec_id_2
rec-0-org   rec-3741-dup-0       0.0        0.0        0.0           0
            rec-3741-org         0.0        0.0        0.0           0
rec-10-org  rec-2162-org         1.0        0.0        0.0           0
            rec-343-org          1.0        0.0        0.0           0
            rec-956-org          1.0        0.0        0.0           0
...                              ...        ...        ...         ...
rec-998-org rec-345-org          1.0        0.0        0.0           0
            rec-3803-org         1.0        0.0        0.0           0
            rec-3936-org         1.0        0.0        0.0           0
            rec-622-org          1.0        0.0        0.0           0
rec-999-org rec-3297-org         1.0        0.0        0.0           0

                             date_of_birth  suburb  state
rec_id_1    rec_id_2
rec-0-org   rec-3741-dup-0               1       0      0
            rec-3741-org                 1       0      0
rec-10-org  rec-2162-org                 0       0      0
            rec-343-org                  0       0      0
            rec-956-org                  0       0      0
...                                    ...     ...    ...
rec-998-org rec-345-org                  0       0      1
            rec-3803-org                 0       0      1
            rec-3936-org                 0       0      0
            rec-622-org                  0       0      0
rec-999-org rec-3297-org                 0       0      0

[50929 rows x 7 columns]
```

1.11.4 Classification

We train our ML models in order to acquire knowledge to classify records in the test set, considering the weights of the attributes.

Deterministic approach

Naive Bayes

```
# Initialise the NaiveBayesClassifier.
nb = rl.NaiveBayesClassifier()
nb.fit(features, true_links)
```

```
p probability P(Match): 0.03769954250034364

m probabilities P(x_i=1|Match): {'surname': {0.0: 0.19739586485459734, 1.0: 0.8026041351454025}, 'addres

u probabilities P(x_i=1|Non-Match): {'surname': {0.0: 0.006304966409006646, 1.0: 0.9936950335909945}, 'a

weights of features: {'surname': {0.0: 31.3079962761764, 1.0: 0.8076966352996334}, 'address_1': {0.0: 0
```

Expectation-Conditional Maximisation

```
# Initialise the Expectation-Conditional Maximisation classifier.
ecm = rl.ECMClassifier()
ecm.fit(features)
```

```
p probability P(Match): 0.04422886741322979

m probabilities P(x_i=1|Match): {'surname': {0.0: 0.30542655786279255, 1.0: 0.694573442137208}, 'address

u probabilities P(x_i=1|Non-Match): {'surname': {0.0: 3.4736045678538596e-07, 1.0: 0.9999996526395413},

weights of features: {'surname': {0.0: 879278.4322353012, 1.0: 0.6945736834046412}, 'address_1': {0.0: 0
```

1.11.5 Evaluation

1.12 Part 2 : Testing

1.12.1 Dataset description

This data set contains 5000 records (2000 originals and 3000 duplicates), with a maximum of 5 duplicates based on one original record (and a Zipf distribution of duplicate records). Distribution of duplicates: 168 originals records have 5 duplicate records 161 originals records have 4 duplicate records 212 originals records have 3 duplicate records 256 originals records have 2 duplicate records 368 originals records have 1 duplicate record 1835 originals records have no duplicate record.

```
Loading data...
5000 records in dataset A
6538 links between dataset A and A
```

```
               given_name    surname street_number           address_1  \
rec_id
rec-1496-org     mitchell      green             7       wallaby place
rec-552-dup-3      harley   mccarthy           177        pridhamstreet
rec-988-dup-1    madeline      mason            54     hoseason street
rec-1716-dup-1   isabelle        NaN            23       gundulu place
rec-1213-org       taylor   hathaway             7      yuranigh court
...                   ...        ...           ...                 ...
rec-937-org          jack   campbell           169          marr street
rec-1200-dup-0    william   lazaroff            12           leah ylose
rec-1756-org     destynii   bowerman            12    halford crescent
rec-1444-org       gianni     dooley            38   ashburton circuit
rec-993-dup-0        jake  westbrook           231  booroondar a street

                              address_2          suburb postcode state  \
rec_id
rec-1496-org                     delmar       cleveland     2119    sa
rec-552-dup-3                    milton         marsden     3165   nsw
rec-988-dup-1   lakefront retrmnt vlge       granville     4881   nsw
rec-1716-dup-1                currin ga        utakarra     2193    wa
rec-1213-org             brentwood vlge             NaN     4220   nsw
...                                 ...             ...      ...   ...
rec-937-org                    rhosewyn        oakleigh     3356   vic
rec-1200-dup-0                 milwlood          forbes     7256   qld
rec-1756-org                     sutton       nollamara     2431   qld
rec-1444-org             brentwood vlge            ryde     6025   qld
rec-993-dup-0                   jodayne  salisbury east     2074   nsw

               date_of_birth soc_sec_id
rec_id
rec-1496-org        19560409    1804974
rec-552-dup-3       19080419    6089216
rec-988-dup-1       19081128    2185997
rec-1716-dup-1      19921119    4314184
rec-1213-org        19991207    9144092
...                      ...        ...
rec-937-org         19770109    1485686
```

```
rec-1200-dup-0           NaN    8072193
rec-1756-org        19880821    6089424
rec-1444-org        19371212    5854405
rec-993-dup-0       19001115    2330929

[5000 rows x 10 columns]
```

1.12.2 Indexing

```
candidate_links = indexing_dataset(df, None, ['surname','date_of_birth', 'soc_sec_id'])
candidate_links
```

```
MultiIndex([(   'rec-1-org', 'rec-1963-dup-0'),
            (   'rec-1-org',  'rec-567-dup-1'),
            (   'rec-1-org',  'rec-910-dup-1'),
            ('rec-10-dup-0',   'rec-10-dup-2'),
            ('rec-10-dup-1',   'rec-10-dup-0'),
            ('rec-10-dup-1',   'rec-10-dup-2'),
            ('rec-10-dup-1', 'rec-1411-dup-0'),
            ('rec-10-dup-1', 'rec-1411-dup-1'),
            ('rec-10-dup-1', 'rec-1411-dup-2'),
            ('rec-10-dup-1',   'rec-1411-org'),
            ...
            ( 'rec-997-org',   'rec-66-dup-0'),
            ( 'rec-997-org',     'rec-66-org'),
            ( 'rec-997-org',  'rec-724-dup-0'),
            ( 'rec-997-org',  'rec-724-dup-4'),
            ( 'rec-997-org',    'rec-724-org'),
            ( 'rec-997-org',    'rec-849-org'),
            ( 'rec-997-org',  'rec-997-dup-0'),
            ( 'rec-999-org',    'rec-234-org'),
            ( 'rec-999-org',  'rec-679-dup-1'),
            ( 'rec-999-org',    'rec-679-org')],
           names=['rec_id_1', 'rec_id_2'], length=40470)
```

1.12.3 Comparison

```
                              surname  address_1  address_2  given_name  \
rec_id_1     rec_id_2
rec-1-org    rec-1963-dup-0       1.0        0.0        0.0           0
             rec-567-dup-1        1.0        0.0        0.0           0
             rec-910-dup-1        1.0        0.0        0.0           0
rec-10-dup-0 rec-10-dup-2         0.0        0.0        1.0           0
rec-10-dup-1 rec-10-dup-0         0.0        0.0        1.0           0
...                               ...        ...        ...         ...
rec-997-org  rec-849-org          1.0        0.0        0.0           0
             rec-997-dup-0        1.0        1.0        1.0           0
rec-999-org  rec-234-org          1.0        0.0        0.0           0
             rec-679-dup-1        1.0        0.0        0.0           0
             rec-679-org          1.0        0.0        0.0           0

                              date_of_birth  suburb  state
rec_id_1     rec_id_2
rec-1-org    rec-1963-dup-0               0       0      0
             rec-567-dup-1                0       0      1
             rec-910-dup-1                0       0      1
rec-10-dup-0 rec-10-dup-2                 1       1      1
rec-10-dup-1 rec-10-dup-0                 1       1      1
...                                     ...     ...    ...
rec-997-org  rec-849-org                  0       0      1
             rec-997-dup-0                1       1      1
rec-999-org  rec-234-org                  0       0      0
             rec-679-dup-1                0       0      0
             rec-679-org                  0       0      0

[40470 rows x 7 columns]
```

1.12.4 Classification

Deterministic approach We consider a match all the pairs that agree on at least 4 attributes.

```
7.0      969
6.0     2066
5.0     1955
4.0     1007
3.0      451
2.0     6929
1.0    27093
dtype: int64
```

```
                              surname  address_1  address_2  given_name  \
rec_id_1      rec_id_2
rec-10-dup-0  rec-10-dup-2        0.0        0.0        1.0           0
rec-10-dup-1  rec-10-dup-0        0.0        0.0        1.0           0
              rec-10-dup-2        1.0        0.0        1.0           0
rec-10-org    rec-10-dup-0        0.0        1.0        1.0           0
              rec-10-dup-1        1.0        0.0        1.0           0
...                               ...        ...        ...         ...
rec-995-dup-0 rec-995-org         1.0        1.0        1.0           1
rec-996-dup-1 rec-996-dup-0       1.0        1.0        1.0           0
rec-996-org   rec-996-dup-0       1.0        1.0        1.0           0
              rec-996-dup-1       1.0        1.0        1.0           1
rec-997-org   rec-997-dup-0       1.0        1.0        1.0           0

                              date_of_birth  suburb  state
rec_id_1      rec_id_2
rec-10-dup-0  rec-10-dup-2                1       1      1
rec-10-dup-1  rec-10-dup-0                1       1      1
              rec-10-dup-2                1       1      1
rec-10-org    rec-10-dup-0                1       1      1
              rec-10-dup-1                1       1      1
...                                     ...     ...    ...
rec-995-dup-0 rec-995-org                 1       1      1
rec-996-dup-1 rec-996-dup-0               1       0      1
rec-996-org   rec-996-dup-0               1       1      1
              rec-996-dup-1               1       0      1
rec-997-org   rec-997-dup-0               1       1      1

[5997 rows x 7 columns]
```

Naive Bayes

```
Predicted number of links: 6428
```

Expectation-Conditional Maximisation

```
Predicted number of links: 6750
```

1.12.5 Comparison

```
Deterministic approach

Confusion matrix:
 [[ 5997   541]
 [    0 33932]]
Fscore: 0.9568408456322298
Recall: 0.9172529825634751
```

```
Precision: 1.0
```

```
Naive Bayes

Confusion matrix:
 [[ 6364   174]
 [   64 33868]]
Fscore: 0.9816443004781737
Recall: 0.9733863566840012
Precision: 0.9900435594275047
```

```
Expectation-Conditional Maximisation

Confusion matrix:
 [[ 6436   102]
 [  314 33618]]
Fscore: 0.9686935580975315
Recall: 0.9843988987457938
Precision: 0.9534814814814815
```

1.13 Results analysis

We conclude that our approach is viabile as shown by the confusion matrices, however it necessitates the true links, which may not be always available (or may require the manpower to do so).